Performance-Portability: Case Studies of NIM and FV3 Models

Mark Govett, Jacques Middelcoff, Duane Rosenberg, Jim Rosinski, Lynd Stringer, Yonggang Yu

NOAA
Earth System Research Laboratory

Comparison

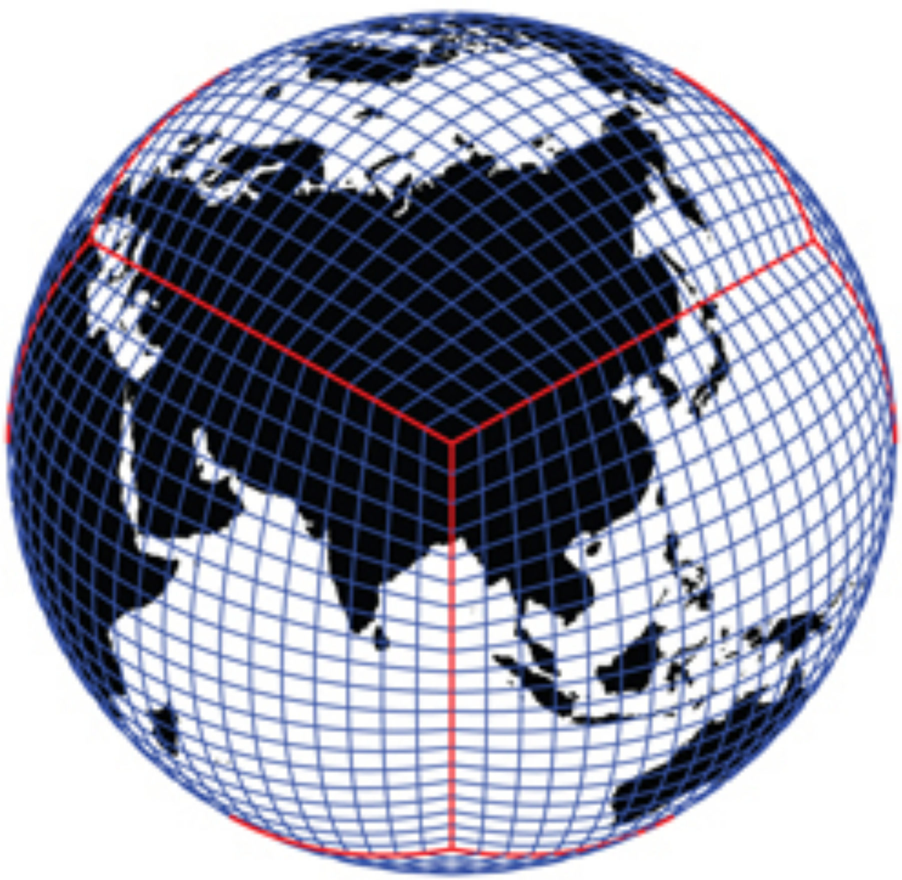
NIM

- Weather Prediction
- Non-hydrostatic
- ~ 4K lines of code
- 2008 – 2015
  - ESRL, Spire Global
  - Designed for GPU, MIC, CPU
- Icosahedral grid
  - All cells treated identically
  - Lookup table for neighbors
- Simple time-step
- Arakawa – A grid
  - All data in cell centers

FV3

- Weather & Climate Prediction
- Hydrostatic, non-hydrostatic
- ~28K lines of code
- 1988 - 2017
  - GFDL, NWS, NASA, NCAR
  - Designed for CPU
- Cube-sphere grid
  - Special cases for edges, corners
  - I – J index for Latitude, Longitude
- Complex time-step
- Arakawa – C & D grid
  - Data in cell centers, edges, corners
  - Transformations between grids

Model Grids

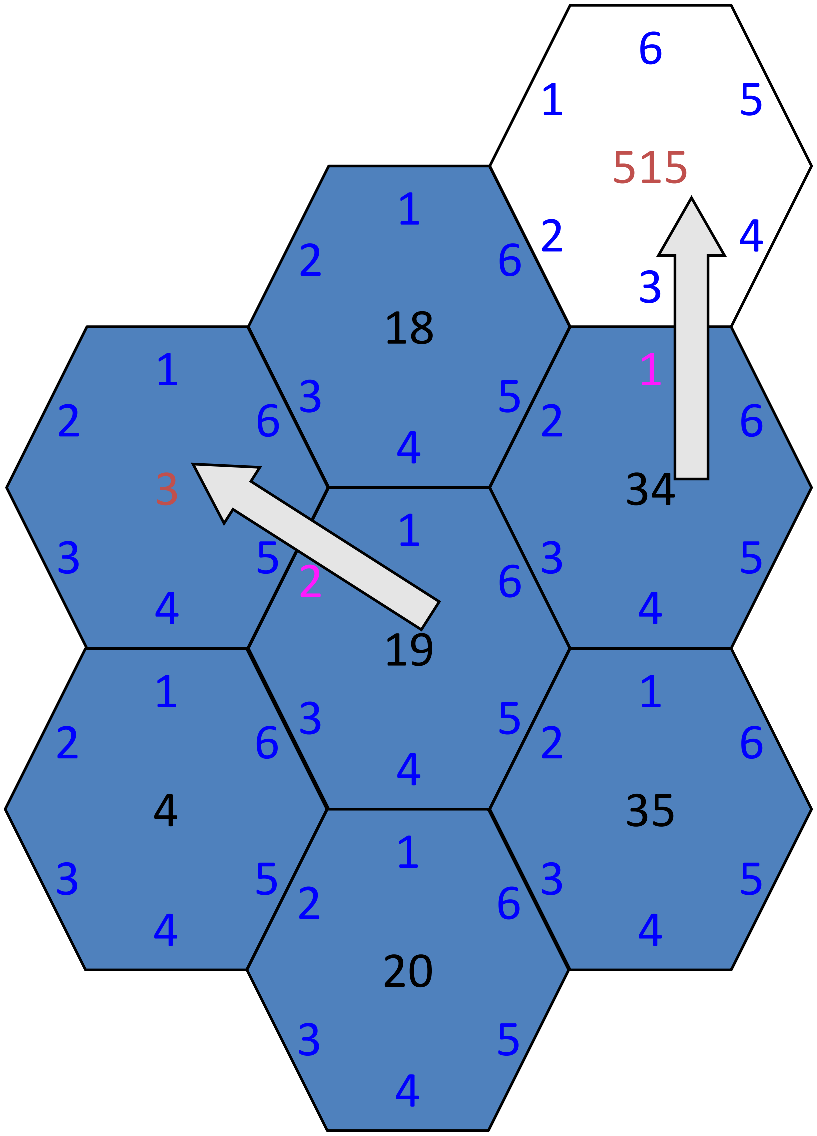


Cube-Sphere Grid
FV3, EndGame, …

Icosahedral Grid
NIM, MPAS, ICON, …

Indirect Addressing Scheme

Used in NIM, Adopted by MPAS



- Single horizontal index
- Store number of sides (5 or 6) in "nprox" array
  - nprox(34) = 6
- Store neighbor indices in "prox" array
  - prox(1,34) = 515
  - prox(2,19) = 3
- Place directly-addressed vertical dimension fastest-varying for speed
- Very compact code
- Indirect addressing costs <1%

(slide courtesy Tom Henderson)

Code Structure & Parallelism

NIM

- Fortran: ~21 routines
- 1-2 deep call tree
- Small routines
- K - I ordering
- OMP, openACC, SMS - MPI

Parallelism

- Vectorization in "K"
  - Except vertical remapping
- Small OMP regions over "I"
  - ~ 100 – 200 lines

10% of peak on Haswell

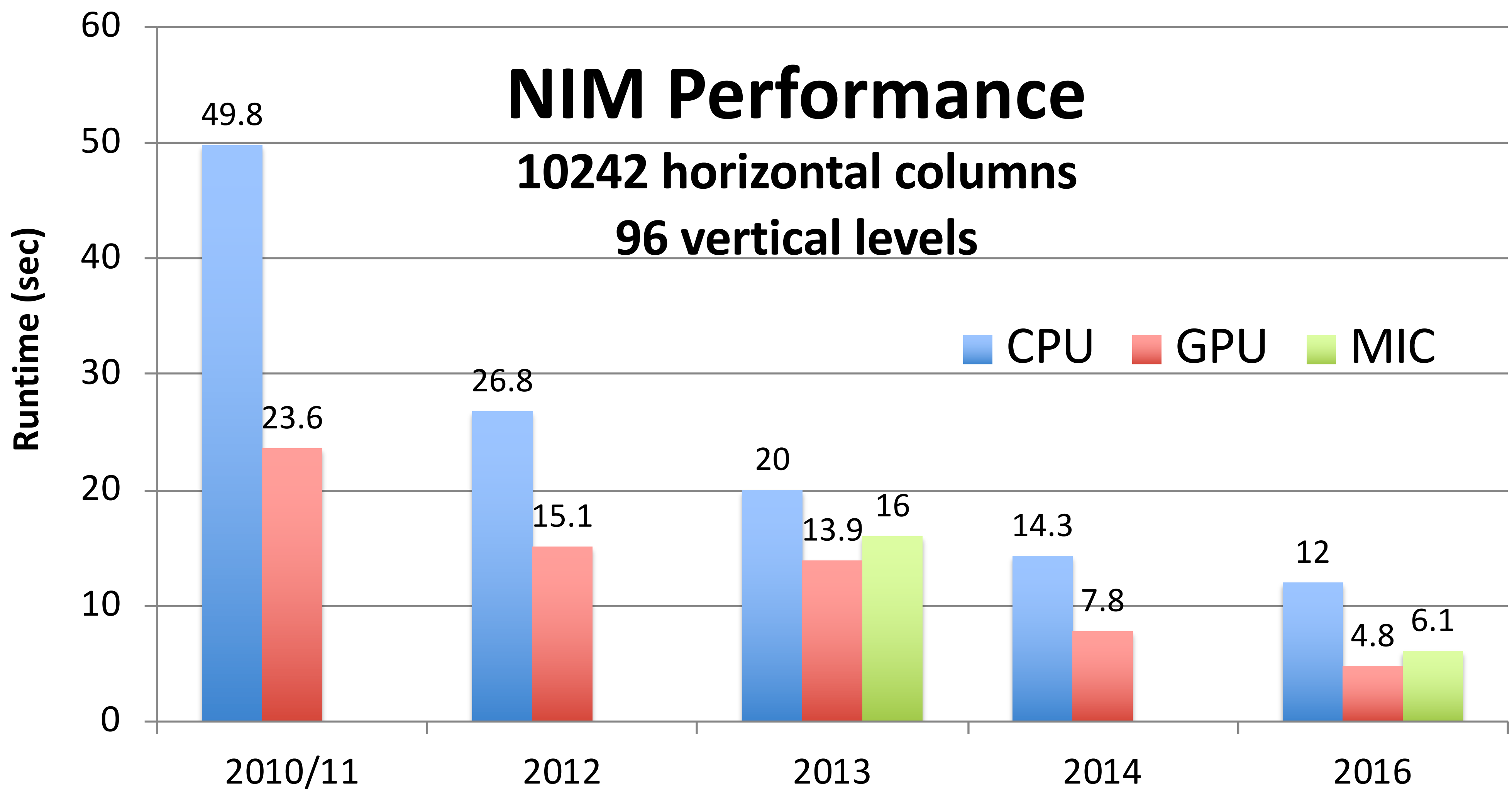
FV3

- Fortran: ~165 routines
- 3-4 deep call tree
- Large routines
- I – J - K ordering
- OMP, openACC, MPI

Parallelism

- Vectorization on "I"or "J"
  - Limited by horiz dependencies
- Large OMP loops over "K"
  - ~1000-5000 lines

10% of peak on Haswell



Year	Intel CPU (cores)	NVIDIA GPU (cores)	Intel MIC (cores)
2010/11	Westmere (12)	Fermi (448)	
2012	SandyBridge (16)	Kepler K20x (2688)	
2013	IvyBridge (20)	Kepler K40 (2880)	Knights Corner (61)
2014	Haswell (24)	Kepler K80 (4992)	
2016	Broadwell (30)	Pascal P100 (3672)	Knights Landing (68)

FV3: Fine-Grain Parallelization

- Increased parallelism needed for GPU
  - Push vertical "k" dimension into routines

Original: I – J

```
do k = 1, npz
  call c_sw(a(:,:,k), )
  call riem_solver( ...
  call update_dz( ...
  call d_sw( ...
enddo


subroutine c_sw (a, )
 real a(isd:ied,jsd:jed)

 do j
  do i
```

Transformed: I – J – K

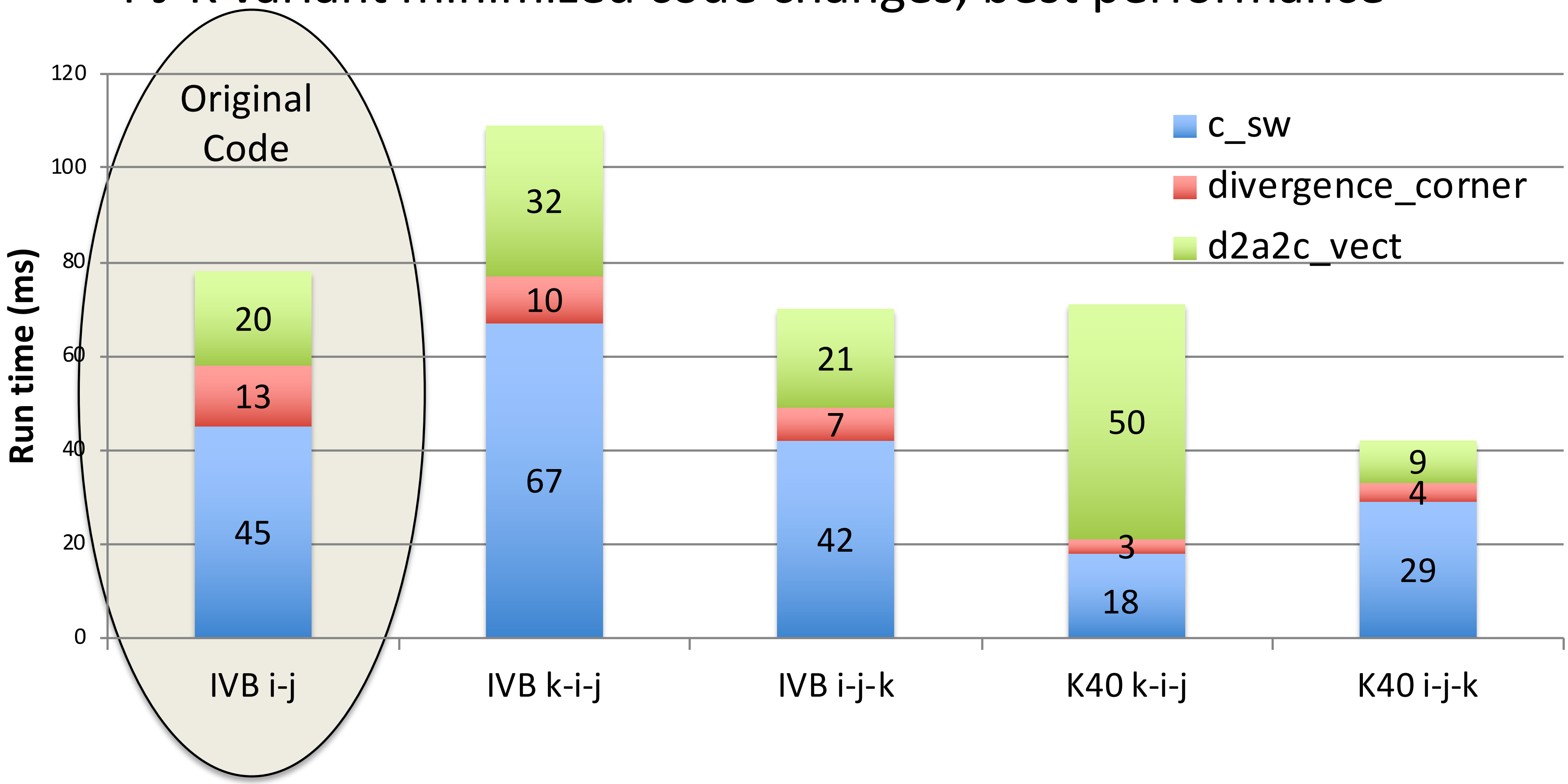
```
call c_sw_3D(a(:,:,:), )
call riem_solver_3D (...
call update_dz_3D ( ...
call d_sw_3D ( ...


subroutine c_sw_3D (a, )
 real a(is:ie,js:je,npz)

 do k
  do j
   do i
```

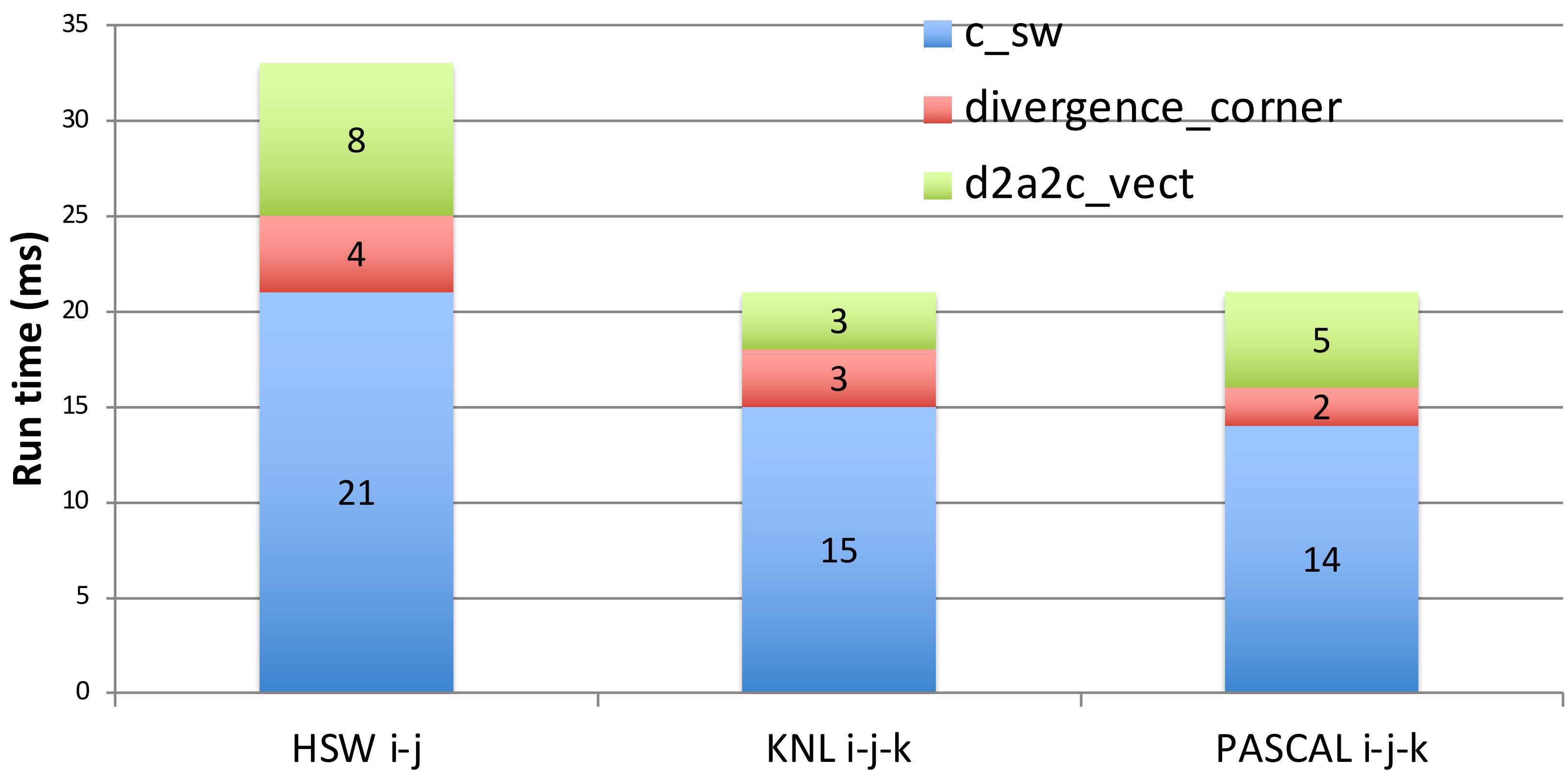
FV3: Shallow Water Kernel

- Compared I-J variant to I-J-K and K-I-J array ordering
- 1.8X faster on the GPU (IVB-20 vs Kepler K40)
  - I-J-K variant minimized code changes, best performance



FV3: Shallow Water Kernel

- 2016 chips: Haswell (24 cores), KNL, Pascal P100
- 1.6X faster on the GPU



Adapting FV3 Dynamics for GPUs

- dyn_core (100%)
  - c_sw (13%)
    - d2a2_vect
    - divergence_corner
  - update_dz_c (2%)
  - riem_solver_c (14%)
  - d_sw (38%)
    - FV_TP_2D (37%)
      - copy_corners (0.1%)
      - xppm (14%)
      - yppm (14%)
    - xtp_v
    - xtp_u
  - update_dz_d (10%)
    - FV_TP_2D (37%)
  - riem_solver (1%)
  - pg_d (5%)
    - nh_p_grad (5%)
  - tracer_2d (6%)
  - remapping (6%)

- Minimize changes to the code
- Require bitwise exact results
- Optimize performance
  - Maintain CPU perf
- Update to latest NWS code periodically
- Work with NWS to merge changes into trunk

FV3 Code Changes

- Push “K” loop in, modify array declarations, remove references to array sections, promote temporaries from 2D to 3D
  - Tens of local variables promoted to 3D
  - Break routine into multiple segments for GPU
    - Decrease register pressure
- Work around compiler bugs, derived types, pointers
- Debugging Challenges
  - Extensive use of pointers and array sections that obfuscate meaning, derived types & openACC
- Performance Issues
  - Promotion to 3D blows out cache
  - Increased number of OMP regions may hurt performance

FV3 Performance

- very preliminary results -

- Full model versus standalone kernel
  - Improved CPU performance over standalone
    - Thread pinning, cache reuse
  - 3D variant runs slower than 2D on CPU
- Haswell CPU, Pascal P100 GPU
  - KNL gave ~15% improvement over 2D code

Routine	2D Intel	3D Intel	3D GPU
C_SW	**0.21**	**0.34**	**0.47**
D2A2_vect	0.31	0.95	0.10
REMAP	1.61	1.55	slower
D_SW	5.32	7.41	slower
FV_TP	0.17	0.26	slower

Performance Portability Takeaways
- Code Design -

- Code simplicity
  - Avoid use of pointers, derived types, abstractions, "new" language constructs
- Memory
  - Registers
    - Small kernels reduce register pressure
  - Shared memory
    - FV3 uses none, NIM used extensively
- Compute
  - Stride-1 essential for vectorization, SIMT, memory access
  - Minimize branching
  - Icosahedral grid treats every cell identically
    - FV3 has special cases for edge, and corner cells
  - Use parallel algorithms, avoid complex algorithms

Conclusion

- Performance portability with single source code was achieved with NIM
  - Design targeted GPU
  - Simple language constructs
  - Maximize parallelism
- Adapting FV3 has been difficult
  - Code changes needed for the GPU, run slower on CPU
  - Still digging into FV3 performance issues & resolution
    - Cache, parallelism, kernel size, memory use
    - Cube-sphere grid
- Collaborative design to focus on fine-grain, portability
  - Development by team of scientists, parallelization experts, computer scientists
  - Use language scientists support / accept